

# Lecture 7: Deep RL

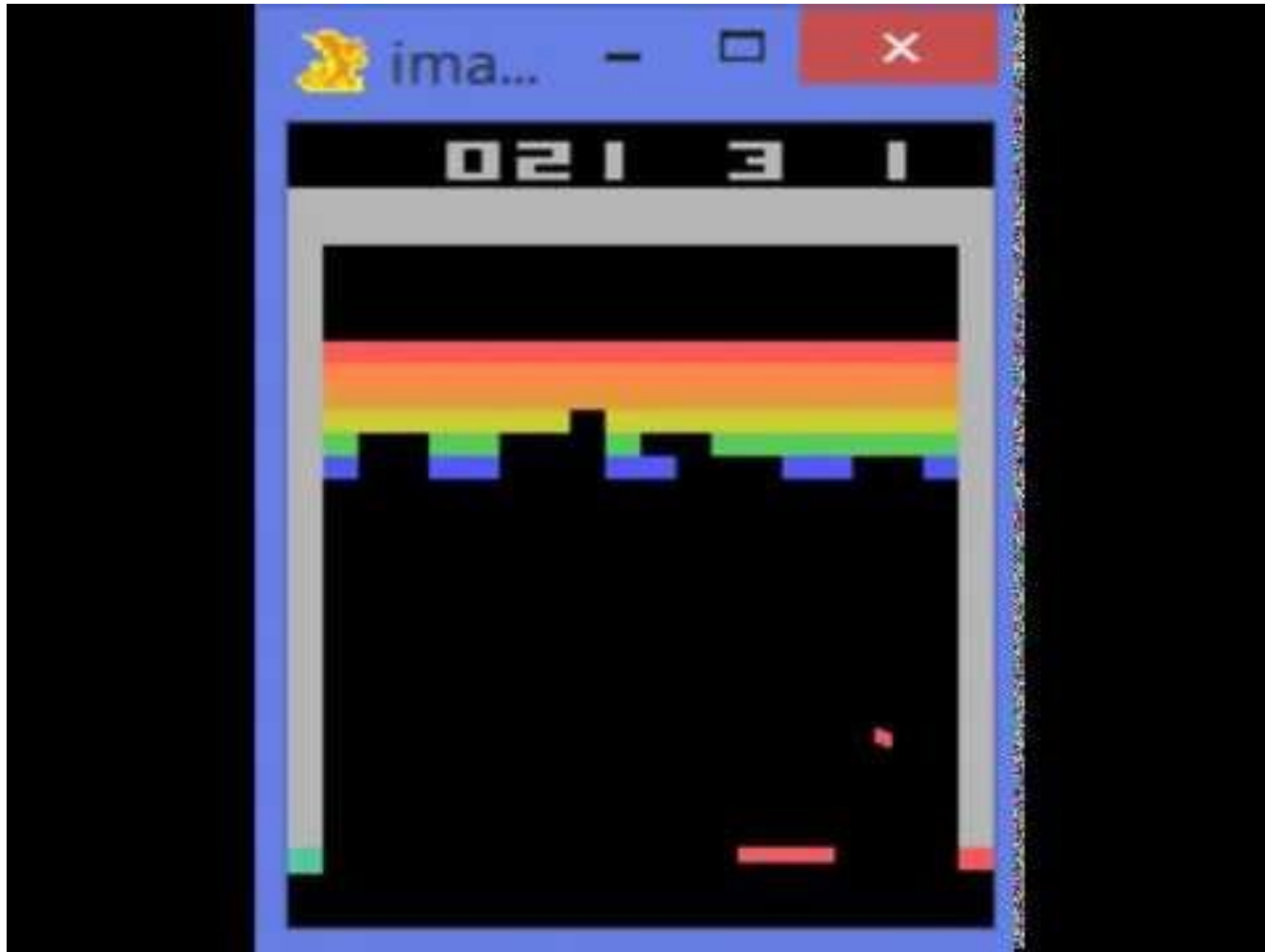
CS234: RL

Emma Brunskill

Spring 2017

Much of the content for this lecture is borrowed from Ruslan Salakhutdinov's class, Rich Sutton's class and David Silver's class on RL.

# Goal: Build RL Agent to Play Atari



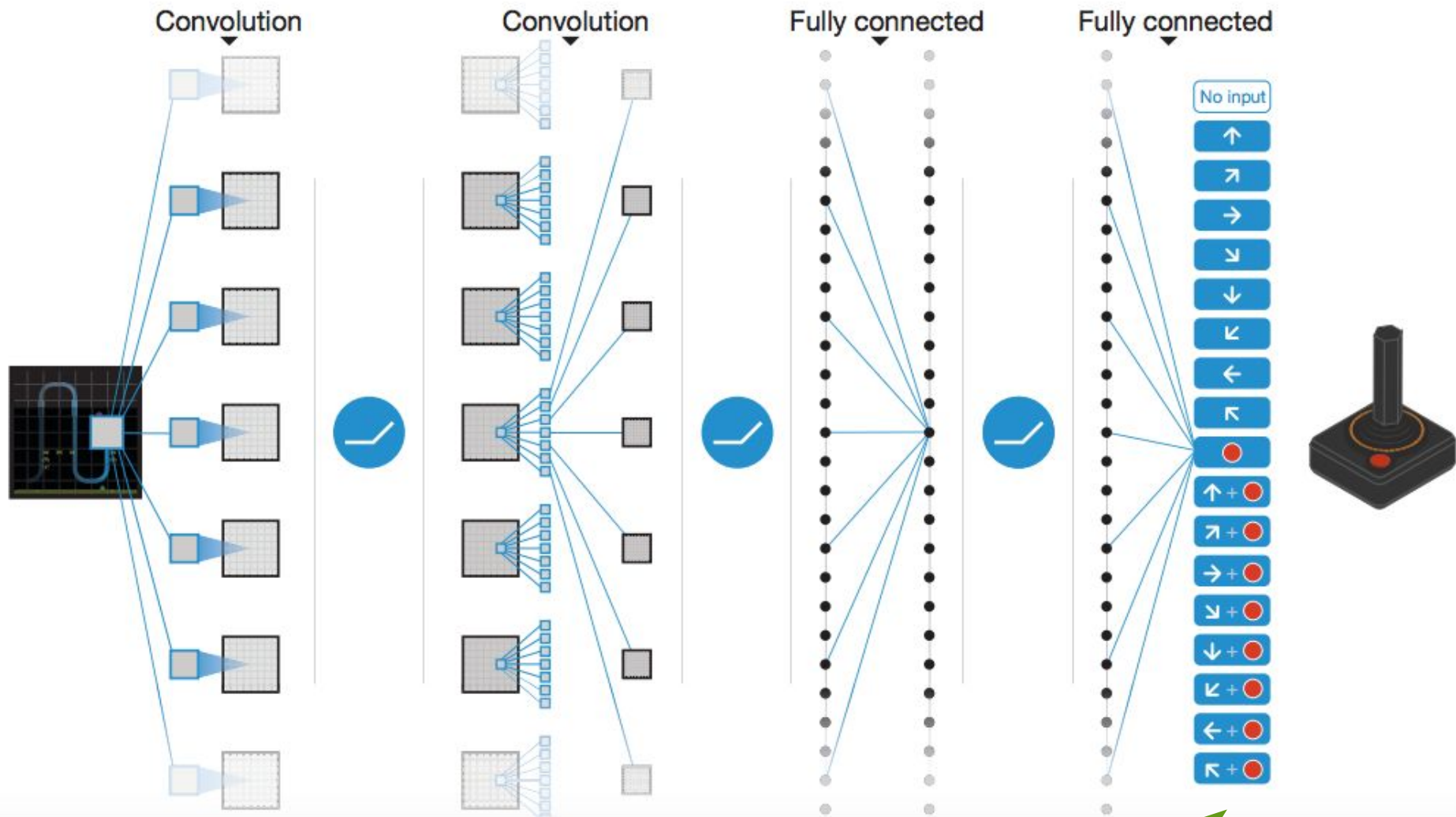
# Generalization in RL

- Need some way to scale to large state spaces
- Important for planning
- Important for learning
- One approach: Model free RL
  - Use value function approximation
  - Discussed using linear weighted combination of features
  - Does this work for Atari?

# Recap: Q-learning + Deep Learning

- Deep Q learning
  - Use deep learning to represent Q function
  - Learns directly from pixels to control

# DQN Architecture



**1 network, outputs Q value for each action**

# Recap: Q-learning + Deep Learning

- Challenge of using function approximation
  - Local updates  $(s, a, r, s')$  highly correlated
  - “Target” (approximation to true value of  $s'$ ) can change quickly and lead to instabilities
- Deep Q-learning
  - Experience replay of mix of prior  $(s_i, a_i, r_i, s_{i+1})$  tuples to update  $Q$
  - Fix target for number of steps

# Recap: DQN

- Experience replay of mix of prior  $(s_i, a_i, r_i, s_{i+1})$  tuples to update  $Q(w)$
- Fix target  $Q(w^-)$  for number of steps, then update
- Optimize MSE between current  $Q$  and  $Q$  target

$$\mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[ \left( \underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}} \right)^2 \right]$$

- Use stochastic gradient descent

# Recap: Double Q-Learning

- Use 2 Q deep nets
- Switch between which network is used as target or for policy selection
- Significant improvement



# Deep RL

- Hugely expanding area
- Will discuss more later in course
- Today: 2 other influential model-free deep RL ideas

# Which Aspects of DQN Were Important for Success?

Game	Linear	Deep netowrk	DQN with fixed Q	DQN with replay	DQN with replay and fixed Q
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Sequest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089

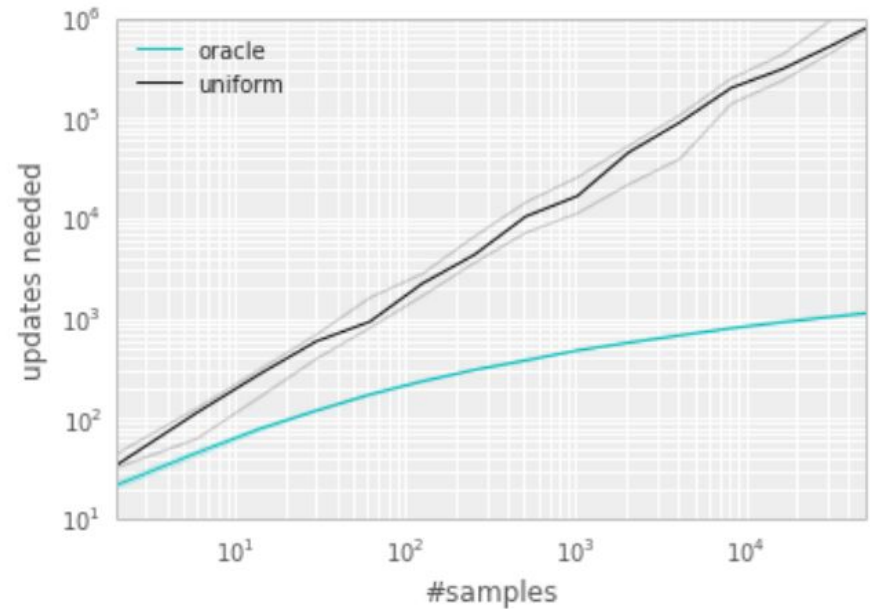
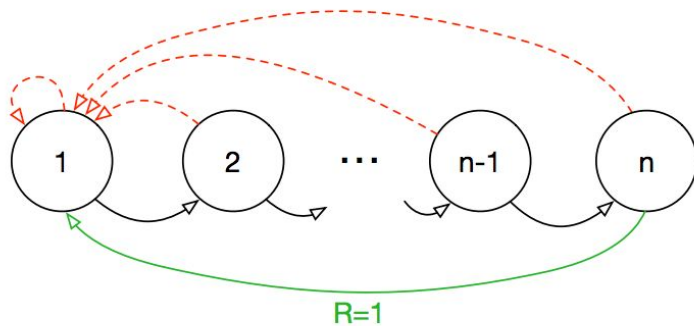
- Replay is **hugely** important

# Order of Replay?

S1	S2	S3	S4	S5	S6	S7
Okay Field Site +1						Fantastic Field Site +10

- In tabular TD-learning, discussed replay could help speed learning
- Repeating some updates seem to better propagate info than others
- Systematic ways to prioritize updates?

# How Much Might Ordering Updates Help?



- Oracle: picks  $(s,a,r,s')$  tuple that will minimize global loss
- **Exponential improvement in convergence!**
- Number of updates needed to converge
- Not practical but illustrates potential impact of order

# Prioritized Experience Replay

- Sample  $(s, a, r, s')$  tuple for update using priority
- Priority of a tuple is proportional to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right|$$

• **Stochastic Prioritization**

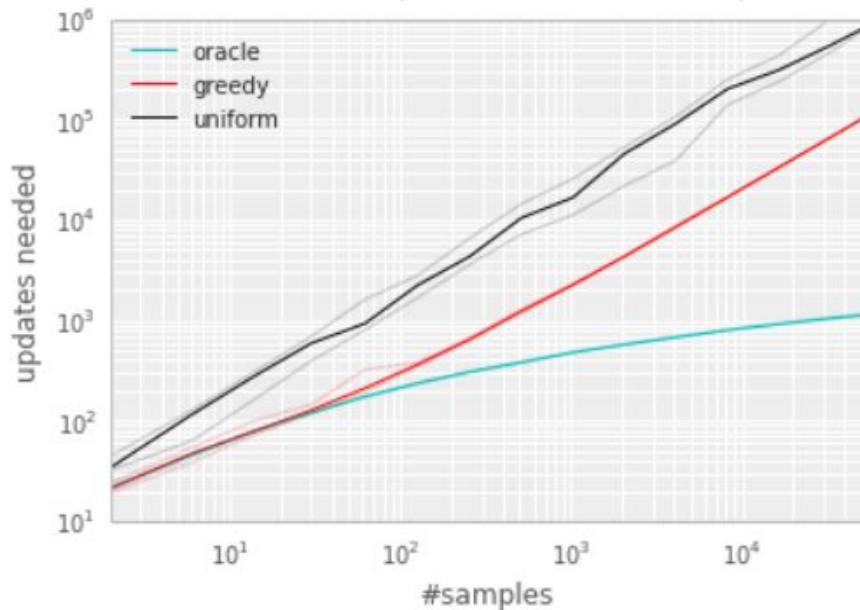
$p_i$  is proportional to  
DQN error

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

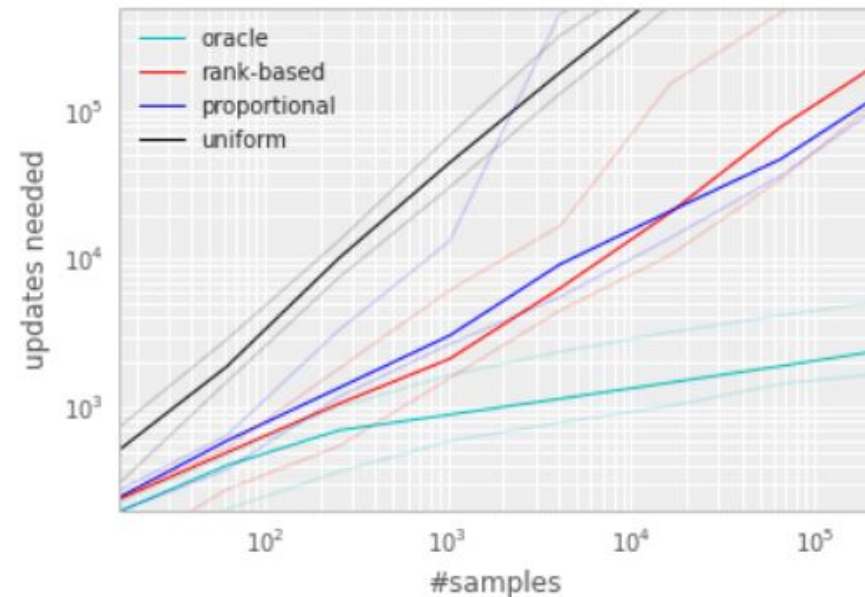
- $\alpha=0$ , uniform
- Update  $p_i$  every update
- $p_i$  for new tuples set to 0

# Impact of Order

## Lookup Table Rep

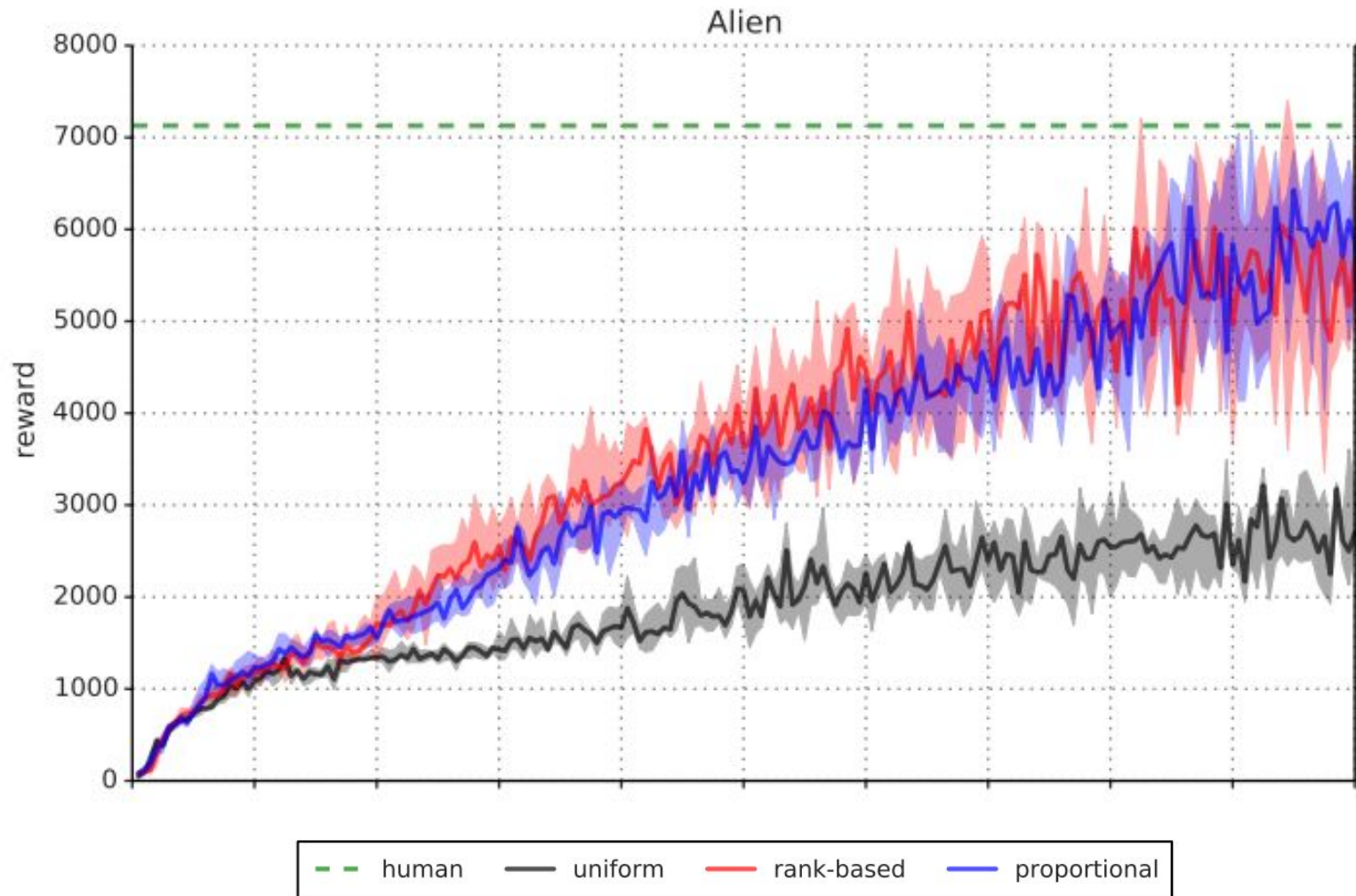


## Linear VFA



- Note: prioritized replay changes distribution of data sampled, which introduces bias
- Can correct for this, see paper

# Substantially Improved Performance



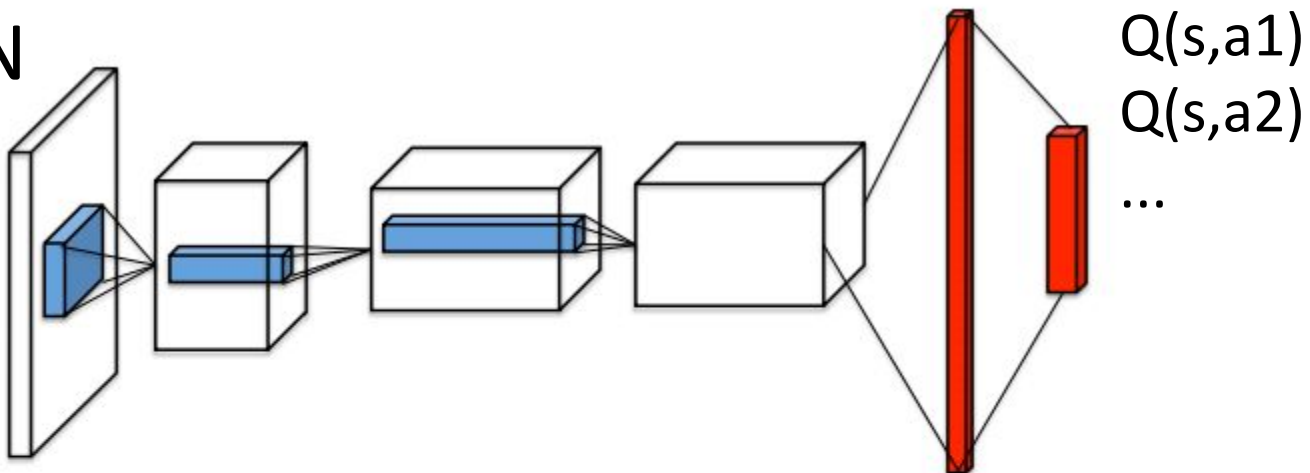
# Value & Advantage Function

- Intuition: Features need to pay attention to determine value may be different than those need to determine action benefit
- E.g.
  - Game score may be relevant to predicting  $V(s)$
  - But not necessarily in indicating relative action values
- Advantage function (Baird 1993)

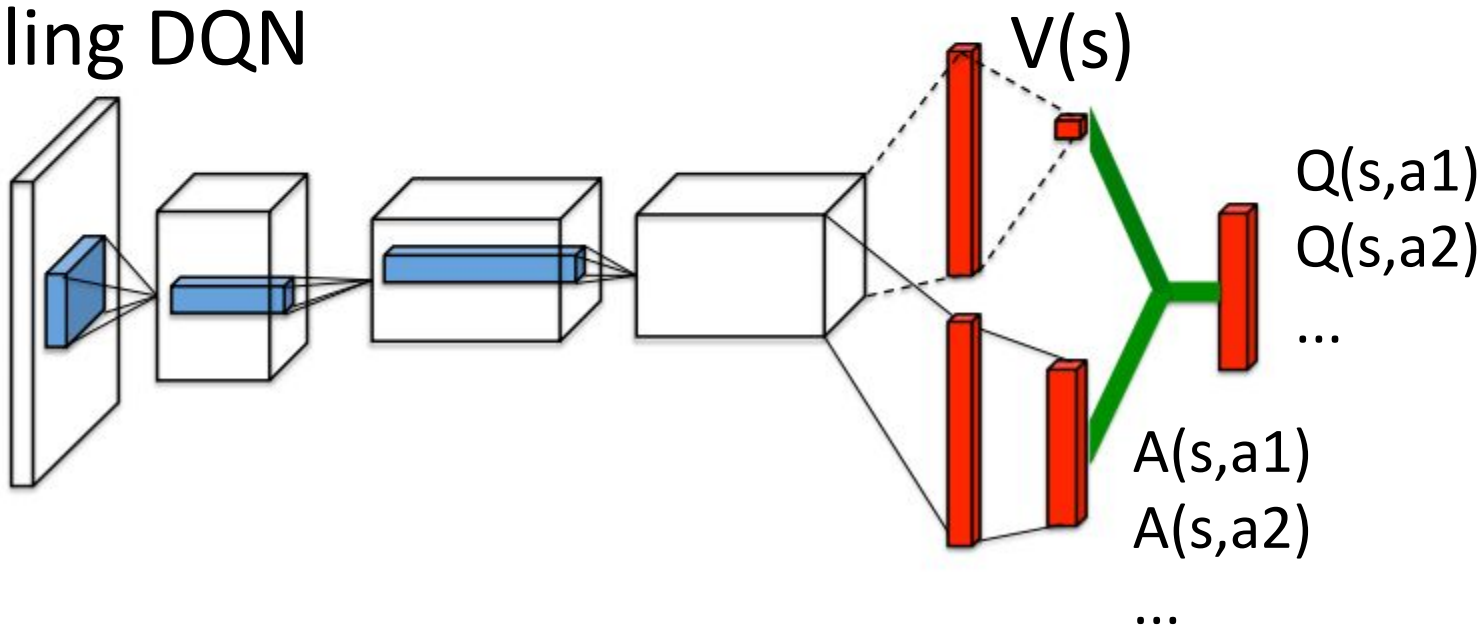
$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$



## DQN



## Dueling DQN



# Identifiability

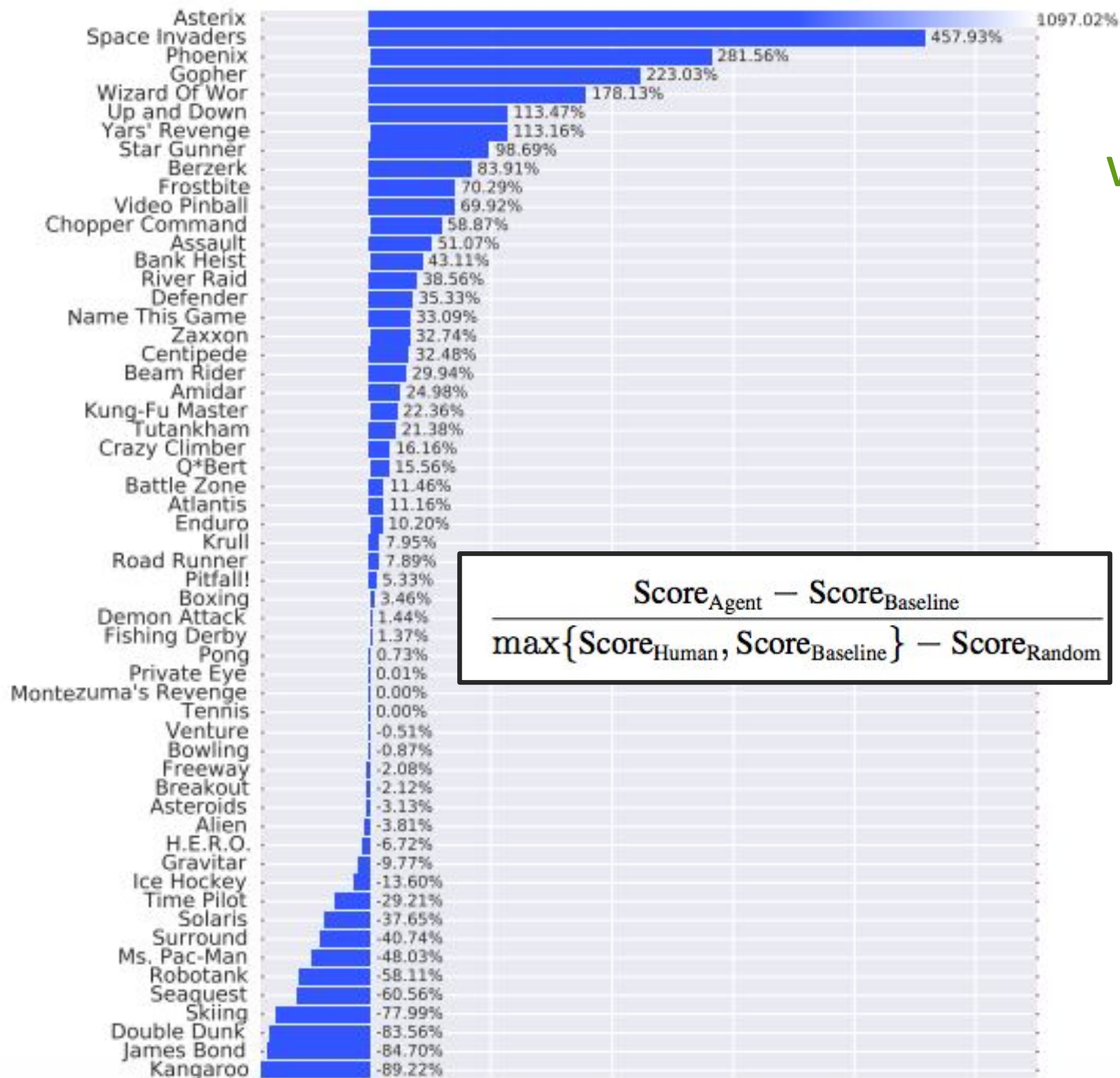
$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

- Unidentifiable
- Option 1: Force  $A(s, a) = 0$  if  $a$  is action taken

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left( A(s, a; \theta, \alpha) - \max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha) \right)$$

- Option 2: Use mean as baseline (more stable)

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left( A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right)$$



$$\frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Baseline}}}{\max\{\text{Score}_{\text{Human}}, \text{Score}_{\text{Baseline}}\} - \text{Score}_{\text{Random}}}$$

Vs DDQN  
w/Prioritized  
Replay

# Model Free Deep RL: Quick Summary

- Stabilize target (proxy for true reward)
- Reuse prior experience in prioritized way
- Separate value and advantage

# Practical Tips for DQN on Atari

(from J Schulman)

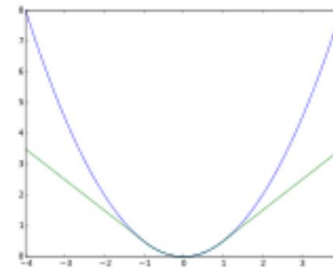
- DQN is more reliable on some Atari tasks than others. Pong is a reliable task: if it doesn't achieve good scores, something is wrong
- Large replay buffers improve robustness of DQN, and memory efficiency is key.
  - Use uint8 images, don't duplicate data
- Be patient. DQN converges slowly—for ATARI it's often necessary to wait for 10-40M frames (couple of hours to a day of training on GPU) to see results significantly better than random policy
- In our Stanford class: Debug implementation on small test environment

# Practical Tips II

(from J Schulman)

- Try Huber loss on Bellman error

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \leq \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$



- Consider trying Double DQN—significant improvement from 3-line change in Tensorflow.
- To test out your data preprocessing, try your own skills at navigating the environment based on processed frames
- Always run at least two different seeds when experimenting
- Learning rate scheduling is beneficial. Try high learning rates in initial exploration period.
- Try non-standard exploration schedules

# Return to Model Free RL...

## Challenges of Target

$$\mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[ \left( \underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}} \right)^2 \right]$$

- Running stochastic gradient descent
- Ideally what should Q-learning target be?

# Return to Model Free RL...

## Challenges of Target

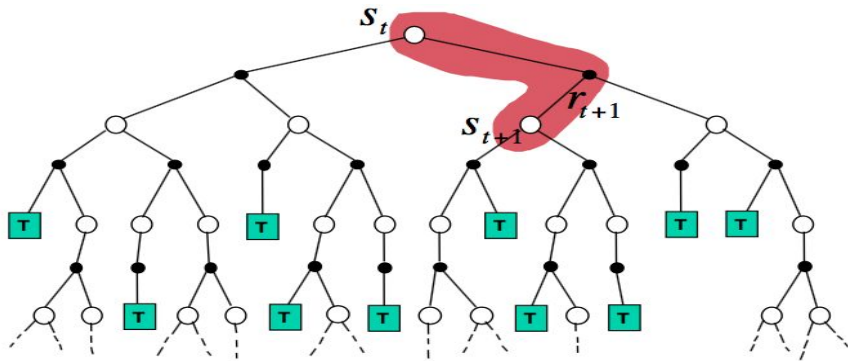
$$\mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[ \left( \underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}} \right)^2 \right]$$

- Running stochastic gradient descent
- Ideally what should Q-learning target be?
  - $Q(s,a)$
  - But we don't know that
  - Could be use Monte Carlo estimate (sum of rewards to the end of the episode)

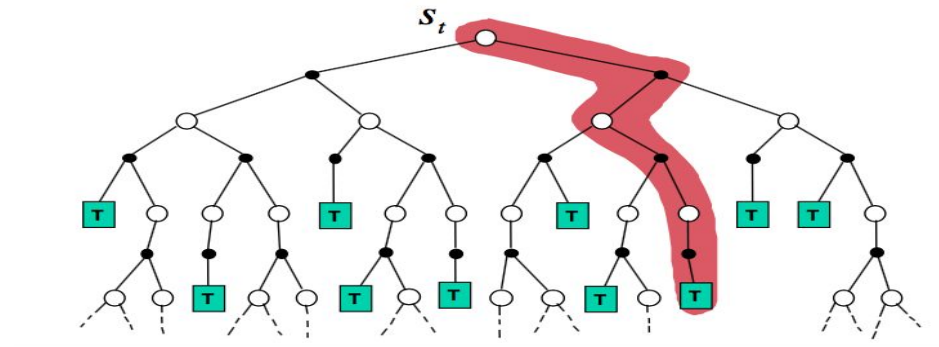


# TD vs Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



# Monte Carlo vs TD Learning

- Computational complexity
- Memory requirements
- Convergence & Q representation
  - Convergence guaranteed?
  - Performance of convergence point?
  - Rate of convergence?
- In on policy case?
  - When evaluating the value of a fixed policy
- In off policy case?
  - When using data to evaluate value of a different  $\pi$

# Monte Carlo vs TD Learning: Convergence in On Policy Case

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in \mathcal{S}} d(s) \left( V^\pi(s) - \tilde{V}^\pi(s, w) \right)^2$$

- where
  - $d(s)$  is generally the on-policy  $\pi$  stationary distrib
  - $\tilde{V}(s, w)$  is the value function approximation

# Monte Carlo Convergence: Linear VFA

(\*Note: correction from prior slides)

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in S} d(s) \left( V^\pi(s) - \tilde{V}^\pi(s, w) \right)^2$$

- where
  - $d(s)$  is generally the on-policy  $\pi$  stationary distrib
  - $\tilde{V}(s, w)$  is the value function approximation
- Linear VFA:  $V(s) = \sum_i w_i f_i(s)$
- **Monte Carlo converges to min MSE possible!**

$$MSVE(w_{MC}) = \min_w \sum_{s \in S} d(s) \left( V^\pi(s) - \tilde{V}^\pi(s, w) \right)^2$$

# TD Learning Convergence: Linear VFA

(\*Note: correction from prior slides)

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in S} d(s) \left( V^\pi(s) - \tilde{V}^\pi(s, w) \right)^2$$

- where
  - $d(s)$  is generally the on-policy  $\pi$  stationary distrib
  - $\tilde{V}(s, w)$  is the value function approximation
- Linear VFA:  $V(s) = \sum w_i f_i(s)$
- **TD converges to constant factor of best MSE**

$$\begin{aligned} MSVE(w_{TD}) &= \frac{1}{1 - \gamma} \min_w \sum_{s \in S} d(s) \left( V^\pi(s) - \tilde{V}^\pi(s, w) \right)^2 \\ &= \frac{1}{1 - \gamma} MSVE(w_{MC}) \end{aligned}$$

# TD Learning vs Monte Carlo: Linear VFA Convergence Point

- Linear VFA:  $V(s) = \sum_i w_i f_i(s)$
- Monte Carlo estimate:
- $MSVE(w_{MC}) = \min_w \sum_{s \in S} d(s) \left( V^\pi(s) - \tilde{V}^\pi(s, w) \right)^2$
- TD converges to constant factor of best MSE

$$MSVE(w_{TD}) = \frac{1}{1 - \gamma} MSVE(w_{MC})$$

- In look up table case what does this say about MSVE of MC and TD?

# TD Learning vs Monte Carlo: Convergence Rate

- Which converges faster?
- Not (to my knowledge) definitively understood
- Practically TD learning often converges faster to its fixed point

# TD Learning vs Monte Carlo:

Finite Data, Lookup Table, Which is Preferable?

- MC Estimates
- vs TD learning with (infinite) experience replay



# TD Learning vs Monte Carlo:

## Finite Data, Lookup Table, Which is Preferable?

- MC Estimates
- vs TD learning with (infinite) experience replay
- 8 episodes, all of 1 or 2 steps duration
  - 1st episode: A, 0, B, 0
  - 6 episodes where observe: B, 1
  - 8th episode: B, 0
- Assume discount factor = 1
- What is a good estimate for  $V(B)$ ?

# TD Learning vs Monte Carlo:

## Finite Data, Lookup Table, Which is Preferable?

- MC Estimates
- vs TD learning with (infinite) experience replay
- 8 episodes, all of 1 or 2 steps duration
  - 1st episode: A, 0, B, 0
  - 6 episodes where observe: B, 1
  - 8th episode: B, 0
- Assume discount factor = 1
- What is a good estimate for  $V(B)$ ?
  - Observed total reward of 1 6 times
  - Observed total reward of 0 2 times
  - Reasonable estimate  $V(B) = 6/8 = 3/4$

# TD Learning vs Monte Carlo:

## Finite Data, Lookup Table, Which is Preferable?

- MC Estimates
- vs TD learning with (infinite) experience replay
- 8 episodes, all of 1 or 2 steps duration
  - 1st episode: A, 0, B, 0
  - 6 episodes where observe: B, 1
  - 8th episode: B, 0
- Assume discount factor = 1
- What is a good estimate for  $V(B)$ ?  $\frac{3}{4}$
- What is a good estimate of  $V(A)$ ?
  - What would TD learning w/infinite replay give?
  - What would MC estimate give?

# TD Learning vs Monte Carlo:

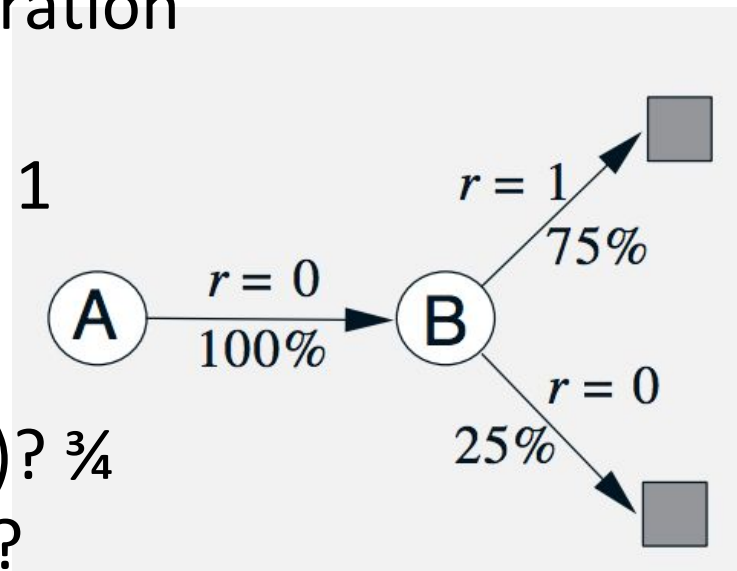
## Finite Data, Lookup Table, Which is Preferable?

- 8 episodes, all of 1 or 2 steps duration
  - 1st episode: A, 0, B, 0
  - 6 episodes where observe: B, 1
  - 8th episode: B, 0
- Assume discount factor = 1
- What is a good estimate for  $V(B)$ ?  $\frac{3}{4}$
- What is a good estimate of  $V(A)$ ?
  - Monte Carlo estimate: 0

# TD Learning vs Monte Carlo:

## Finite Data, Lookup Table, Which is Preferable?

- 8 episodes, all of 1 or 2 steps duration
  - 1st episode: A, 0, B, 0
  - 6 episodes where observe: B, 1
  - 8th episode: B, 0
- Assume discount factor = 1
- What is a good estimate for  $V(B)$ ?  $\frac{3}{4}$
- What is a good estimate of  $V(A)$ ?
  - Monte Carlo estimate: 0
  - TD learning w/infinite replay:  $\frac{3}{4}$ 
    - Computes certainty equivalent MDP
    - MC has 0 error on training set
    - But expect TD to do better-- **leverages Markov structure**



# TD Learning & Monte Carlo: Off Policy

- In Q-learning follow one policy while learning about value of optimal policy
- How do we do this with Monte Carlo estimation?
  - Recall that in MC estimation, just average sum of future rewards from a state
  - Assumes always following same policy

# TD Learning & Monte Carlo: Off Policy

- In Q-learning follow one policy while learning about value of optimal policy
- How do we do this with Monte Carlo estimation?
  - Recall that in MC estimation, just average sum of future rewards from a state
  - Assumes always following same policy
- Solution for off policy MC: Importance Sampling!

# TD Learning & Monte Carlo: Off Policy

- With lookup table representation
  - Both Q-learning and Monte Carlo estimation (with importance sampling) will converge to value of optimal policy
  - Requires mild conditions over behavior policy (e.g. infinitely visiting each state--action pair is one sufficient condition)
- What about with function approximation?



# TD Learning & Monte Carlo: Off Policy

- With lookup table representation
  - Both Q-learning and Monte Carlo estimation (with importance sampling) will converge to value of optimal policy
  - Requires mild conditions over behavior policy (e.g. infinitely visiting each state--action pair is one sufficient condition)
- What about with function approximation?
  - Target update is wrong
  - Distribution of samples is wrong

# TD Learning & Monte Carlo: Off Policy

- With lookup table representation
  - Both Q-learning and Monte Carlo estimation (with importance sampling) will converge to value of optimal policy
  - Requires mild conditions over behavior policy (e.g. infinitely visiting each state--action pair is one sufficient condition)
- What about with function approximation?
  - Q-learning with function approximation can diverge
    - See examples in Chapter 11 (Sutton and Barto)
  - But in practice often does very well

# Summary: What You Should Know

- Deep learning for model-free RL
  - Understand how to implement DQN
  - 2 challenges solving and how it solves them
  - What benefits double DQN and dueling offer
  - Convergence guarantees
- MC vs TD
  - Benefits of TD over MC
  - Benefits of MC over TD